Artificial Intelligence Project---RLE and MIT Computation Center

Memo 10

THE LISP DIFFERENTIATION DEMONSTRATION PROGRAM

by K. Maling

1. Introduction

This program is a byproduct of the machine language which is being developed for the Artificial Intelligence project. It was written because the process of differentiation and to some extent that of simplification, turned out to be very conveniently expressable in LISP. There are two main reasons for this: one is the fact that algebraic expressions are most easily represented in a computer by means of a list language and the other is the ability of LISP to describe recursive processes.

The program is internally in two distinct parts. The first is the differentiation program, which is written out for the case of PLUS and TIMES (The only differentiable functions that do not have a fixed number of arguments) and which handles all other functions by making substitutions of their arguments into the gradient. It is up to the user to add the gradients of functions which he wishes to differentiate. It will normally take one card per function and there is no restriction on the number of functions or their order.

When the differentiation program was first tested out it was found that the resulting expressions were usually enormous and contained a large number of redundant symbols. The second part therefore consists of a simplification program which is written out for PLUS and TIMES, the chief offenders, and for all other functions the recognition of degenerate cases is built it. Again it is up to the user to add the degenerate cases and their equivalents for the functions in which he is interested, and again there are no restrictions on their number or order.

There is one further feature which involves the Universal LISP Function "Apply". Apply is in effect an interpreter which causes any process defined in LISP to be performed on an expression. The user may have his own list processes carried out in two places, (a) during the simplify program, whenever any function appears which he has introduced (any one LISP program will be for one function only, but each function may have such a program), (b) after the above simplification

has been completed he may carry out any further manipulations on the resulting expression. The demonstration program is therefore quite general.

2. [illegible]

Algebraic expressions and LISP expressions are written in a restricted external notation. For instance, we write

x(x+1)sin(y) as (TIMES,x,(PLUS,x,1),(SIN,y))

and cons(car(x),0) as (cons,(car,x),INTV,0))

The output is in a similar form. An infix reader and printer are being considered and may be added at a later date. At the moment only literals and digits are legal characters and the names of functions like RATIO and MINUS must be written as words. Numbers are treated exactly like symbols and "simplify" does no arithmetic with them, nor does it enumerate. The user is cautioned against a common source of error, namely the failure to provide a sufficient number of right parentheses.

3. Differentiation

3.1 For the following sections familiarity with the basic LISP functions and the method of writing program functions is assumed.

Let L be the list representation of an algebraic expression.

Let V be the variable with respect to which it is being differentiated. Then

```
Diff(L,V)=/L=V—>1,  atom(L)—>0,  car(L) = PLUS—>cons(PLUS,maplis
     (cdr(L),λ(J,diff(car(J),V)))),car(L)=TIMES—>cons(PLUS,maplis
     (cdr(L),λ(J,cons(TIMES,maplis(cdr(L),λ(K,J≠K—>car(K),T —>
     diff(car(K),V))))))),T—>cons(PLUS,map2(sublis(pair(car(gradnt
     (car(L))),cdr(L)),cadr(gradnt(car(L)))),cdr(L),λ(J,K,
     list(TIMES,copy(car(J)),diff(car(K),V)))))/
Gradnt(M)= search(M,λ(J,car(J)=gradl),λ(J,cadr(J)),print comment)
```

3.2 The case of functions other than PLUS and TIMES is not easily understood from reading this definition, and an example is therefore given. d(u/v)/dx

d(u/v)dx = $1/v.(dx/dx) + (-u/v^2).(dv/dx)$

The gradient of (RATIO,u,v) is therefore $(1/v,(-u/v^2))$

The following expression is added to the property list of RATIO.

(GRADL,((u,v),((RATIO,1,v),(RATIO,(MINUS,0,u),(TIMES,v,v)))))

Suppose that (RATIO, 1,x) is being differentiated.

```
gradnt(car(L)) = gradnt(RATIO)→((u,v),((RATIO,1,v),(RATIO,
    (MINUS,0,u),(TIMES,v,v)))) = G.
```

Then

```
pair(car(G), cdr(L))→((u,1),(v,x)) = H
```

and

```
cadr(gradnt(car(L)))→((RATIO,1,v),(RATIO,(MINUS,0,u),
    (TIMES,v,v)))=I
```

hence

```
sublis (H,I)→((RATIO,1,x),(RATIO,(MINUS,0,1,(TIMES,x,x)))=M
```

so that

```
map2(M,cdr(L), λ(J,K, list(TIMES,copy(car(J)),diff(car
    (K),V))))→((TIMES,(RATIO,1,x),0),(TIMES,(RATIO,(MINUS,0,1)
        (TIMES,x,x)),1))
```

and when preceded by a PLUS this is the required result, though in some need of simplification.

3.3 It will be noted that MINUS is treated as a binary operator, an arbitrary decision that was prompted by a wish to treat PLUS and MINUS as similarly as possible to TIMES and RATIO.

Since d(u-v)/dx = du/dx - dv/dx

One writes the gradient of MINUS as (1, (MINUS,0,1))

so that on the property list of MINUS should appear

```
    (GRAD1,((u,v), (1,(MINUS,0,1))))
```

However if a unary minus were used, one would put on the property list

```
    (GRAD1,((u),((MINUS,1))))
```

4. Simplification

4.1 Sums and products are simplified in the following ways.

1. Products that contain a zero term are replaced by zero.
2. Zero terms in sums or unit terms in products are deleted.
3. Sums of sums are combined to contain only one PLUS and products of products are combined to contain only one TIMES.
4. A sum of no terms is zero, a product of no terms is one.
5. A sum or products of only one term is the term itself.

Let L be a well-formed algebraic expression in LISP.

```
Simplify (L) =/atom(L)→return(L),
    K=maplis(cdr(L),λ(J,simplify(car(J))))
    car(L)≠PLUS→go(a1)
K1=mapcon(K,λ(J,car(J)=0→Λ,caar(J)=PLUS→cdar(J),T→cons
(car(J),Λ)))
null(K1)→return(0)
null(cdr(K1))→return(car(K1))
T→return(cons(PLUS,K1))
car(L)≠TIMES→go(a2)
```

```
T→search(K,λ(J,car(J)=0),return(0),go(a3))
(a3)        K1=mapcon(K,λ(J,car(J)=1→Λ,caar(J)=TIMES→cdar(J),T→
     cons(car(J),Λ)))
     null(K1)→return(1)
     null(cdr(K1))→return(car(K1))
          T→return(cons(TIMES,K1))
(a2)        M=search(car(L),λ(J,car(J)=SPEC1),λ(J,cadr(J)),go(a4))
return(search(M,λ(J,inst(cadar(J),caar(J),K),λ(J,sublis(inst(cadar
     (J),caar(J),K),caddr(J)),L))
(a4)        return(search(car(L),λ(J,car(J)=SPEC2,λ(J,apply(cadadr
     (J),L,caadr(J))),L)
```

4.2 Examples will again be given to clarify the method whereby degenerate cases are recognized, and whereby the user can introduce his own program.

Suppose (RATIO,x,x) is to be replaced by "1" and (RATIO,x,1) by "x".

Many other cases such as 0/y, x/0, x/(y/z), could of course be added.

We add to the property list of RATIO

(SPEC1,(((u,u),((u),(v)),1),((u,v),((u),(v,1)),u)))

M of (a2) above will be cadr of this expression.

Where L is (RATIO,x,x)

inst(((u),(v)),(u,u),(x,x))→((u,x))

and sublis (((u,x)),1)→1.

Where L is (RATIO,x,1)

inst(((x),(v,1)),(u,v),(u,v),(x,1))→((u,x),(v,1))

and sublis (((u,x),(v,1)),u)→x

An example with "Apply" will be provided later.

5. Input Statements

The imperatives which the program can obey are

(DIFFERENTIATE, (SIN,X),X)

which means, differentiate sin(x) with respect to x.

(SIMPLIFY,(PLUS,Y,0))

The meaning is obvious.

STOP   causes the computer to halt.

To add gradients, degenerate cases or LISP program to the property list of a function, one must write

(FUNCTION,SIN) (GRAD1,((U),((COS,V))))   or

(FUNCTION,LOG) (SPEC1,(((U),((U,1)),0))

It is very important to spell the following words correctly; DIFFERENTIATE, SIMPLIFY, FUNCTION, PLUS, TIMES, STOP, GRAD1, SPEC1, SPEC2.

It is also essential to have the correct depth of parentheses, and to include matching right hand parentheses.

6. Program Print-out

The demonstration will always begin with

"THIS IS A DEMONSTRATION OF SYMBOLIC DIFFERENTIATION USING LISP, A LIST PROCESSING LANGUAGE. THE CARD CONTAINING THE EXPRESSION TO BE DIFFERENTIATED, AND THE GRADIENTS OF ALL THE FUNCTIONS USED IN IT, SHOULD FOLLOW THE DECK WHICH HAS JUST BEEN READ IN.

NOW PRESS THE START KEY.

A card reading (DIFFERENTIATE,(E),x) causes the print-out

(E)

IS NOW BEING DIFFERENTIATED WITH RESPECT TO

X

The result is presented by

THE DERIVATIVE IS

E'

A card reading (SIMPLIFY,(E_1)) causes the print out

(E_1)

CAN BE SIMPLIFIED TO

(E_2)

A card reading (FUNCTION,G) (GRAD1, (E)) causes the print out

G

HAS NOW ON ITS PROPERTY LIST

(GRAD1,(E))

If the program is called upon to differentiate a function without having previously been provided with the gradient, then the program will print out:

THE PROGRAM HAS NOT BEEN GIVEN THE GRADIENT OF

F

PUT A CARD ON WHICH ITS GRADIENT IS PUNCHED IN THE CARD READER AND PRESS THE START KEY.

If the program is given an expression which is not of the types described in section 5, it will print.

E

IS NOT AN ACCEPTABLE EXPRESSION. TRY AGAIN.

If the program attempts to read when there are no more cards in

the card reader, either because the user has pressed the start key after a machine halt or because the program has just read an expression of the form (FUNCTION,G) then the program will print.

THERE IS NO CARD FOR THE READER TO READ. TRY AGAIN.

7. Checking Out Expressions

It was mentioned in Section 1 that the user can add any LISP program of his own, which will be executed after the simplify program described in Section 4 has been executed, and will use as its argument the result of such previous simplification. The intention of this facility is to make possible cancellation, counting, small integer arithmetic and infix printing, for example.

The user is referred to the LISP programmer's Manual for information on checking out programs for Apply. This LISP program can then be added to the demonstration program by writing

(FUNCTION, SIMPLIFY) (SPEC2,((A),(F)))

Both SIMPLIFY and DIFFERENTIATE (unsimplified) are available to Apply as subroutines. The purpose of the imperative

(SIMPLIFY,E)

is of course to make easy the checking out SPEC1 and SPEC2 type additions to the property list of functions. Note that nothing added to the property lists of PLUS and TIMES will affect the program in any way.

The authors of this program accept no responsibility for any results produced by it, right or wrong, and do not wish to answer any questions about it.